Available online at www.sciencedirect.com

ScienceDirect

Procedia Computer Science 46 (2015) 774 – 783

Procedia Computer Science

International Conference on Information and Communication Technologies (ICICT 2014)

A VDM-based Approach for Specifying and Testing Requirements of Web-Applications

Souvik Sengupta[a*] , Ranjan Dasgupta[b]

[a]*Bengal Institute of Technology, Kolkata, India*
[b]*NITTTR Kolkata, India*

Abstract

The objective of this paper is to demonstrate the use of formal methods in a uniform way for functional as well as interface-requirements. We propose some add-ons to 'Vienna Development Method Specification Language'syntaxes to cover the interface-requirements of a web-based application. We also propose a framework to support the transformation of the conventional SRS to a design specification, and a Finite State Machine based verification model, to test the design specification against the SRS.

Peer-review under responsibility of organizing committee of the International Conference on Information and Communication Technologies (ICICT 2014)

Keywords: Formal Requirement Specifications; VDM-SL;Interface-requirement; Model Based Testing

1. Introduction

Conventional software requirements are primarily classified as functional and non-functional requirements. Functional Requirements (FR) define the actions that must take place in the software in accepting and processing the inputs and generating the outputs with or without support from the library functions, standard database etc.[9]. FR are

* Corresponding author. Tel.: +91-033-23458004
*E-mail address:*mesouvik@hotmail.com

Peer-review under responsibility of organizing committee of the International Conference on Information and Communication Technologies (ICICT 2014)
doi:10.1016/j.procs.2015.02.146

easier to specify using formal models because such requirements can be broken down into a sequence of sub requirements and each such sub requirements can be expressed in some mathematical and/or logical form which is also easier to code and test. On the other hand, interface-requirements, being Non-Functional Requirements (NFR), are hard to define in a formal way. The case of web-applications is even more complex due to the event driven nature of such applications. The sequencing of the activities, representing the requirements, cannot be predefined as their (activities) order of occurrence in the system is determined by the users of the web-application. Moreover, some of the interface-requirements can appear as an integral part of the FR. For example, clients' need of a dropdown list or Ajax based suggestion in a text box is inseparably attached with the input method of a functional requirement. This type of compound requirements (interface-requirement(s) attached with a functional requirement) often has the problem of stating different parts of a requirement at different levels of abstraction. The interface-requirements are often specified with more details than the FR are. Consequently, we often use different specification languages for different types of requirements but then the interface-requirements, if handled in isolation, can lead to traceability problem for the developers, especially in web- application[1]. The Vienna Development Method Specification Language (VDM-SL) is a well-established formalizing tool for Software Requirements Specification (SRS) but does not have much built-in support in terms of data types and operations to handle complexity of web-based application like hyperlink navigation, client-server communication, and event driven approach. We propose some add-ons to the VDM-SL so that it can be used to model FR and NFR of a web-application in an integrated way.

In conventional software engineering approach, Unified Modeling Language (UML), Entity Relationship Diagram (ERD), and Data Flow Diagram (DFD) are used to specify the requirements in the SRS as well as in the design specification. In the SRS, these diagrams represent system's behavior in a higher level of abstraction, whereas in the design specification they are used for a more detailed description of the system. The rich and generic nature of these models (UML/ERD/DFD) gives us the liberty to make specifications at different levels of abstraction. In contrast, though in recent years VDM (a common platform for VDM-SL, VDM++, VDM-RT) has experienced a paradigm shift from a definition language to a development method[4], still VDM is more useful in requirement specification and requirement analysis rather than in design specification. Moreover, its features are inadequate to model compound requirements that have parts stated at different levels of abstraction. The difference of level of abstractions in requirements produces a semantic gap between an informal specification of the requirements and an implementable design specification. In order to bridge the gap, we introduce a requirement analysis model with help of VDM-SL specifications for two different abstraction-levels, namely requirements-specification-level (level-1) and design-specification-level (level-2). We also propose a technique to convert a level-1 specification in to a level-2 specification. Finally, we have also proposed a technique to create a Finite State Machine (FSM) model from the level-2 specification and used it to verify the design specification (level-2 VDM-SL) against the SRS.

The remainder of the paper is organized as follows: The next section describes the related works. Section 3 illustrates the proposed framework and in section 4, we discuss the perspective of formalizing interface-requirements in web-applications. Next, in section 5, we propose the add-ons to VDM-SL at two different levels. Section 6 and 7 introduce the conversion techniques. Section 8 explores a case study and finally section 9 is the conclusion.

2. Related work

The work presented in this paper has legacy from three different disciplines, i) writing formal requirement specifications for web-applications using VDM-SL, ii) using formal methods for interface-requirements and iii) applying model based testing for web based application using FSM.

The idea of extending the VDM-SL to support web-applications can be broadly seen as an approach towards Domain-specific languages (DSLs). Marjan Mernik et. al.[14] tried to aid the DSL developer, by providing a systematic survey on the patterns in the decision, analysis, design, and implementation phases of DSL development.

Richard Atterer[7] presented a conversion model for requirements from high-level definition to low-level definition through reification being done by VDM-SL. It is observed that VDM-SL is particularly suitable where a) operations cannot be explicitly specified, b) the structure of the system is not modular, and c) the features are at higher level of abstraction in the specification. In another approach of formal requirement analysis based on data flow analysis and

rapid prototyping, by Shaoying Liu [8], some of the limitations of VDM-SL as a specification tool are identified. This includes a) lack of data types to handle complex applications (specially web-application),b) pre-post condition based specification against operational constraints c) lack in modeling concurrent execution, and iv) lack in event driven modeling.

Jeff Offutt, in his work,[5] discussed three aspects of web applications that are very useful for creating FSM model, (1) an extremely loose form of coupling that features distributed integration, (2) the ability of the users to change the potential flow of execution directly, and (3) the dynamic creation of HTML forms. This model is based on the notion of atomic sections, which are the logical part of a web page. This paper represented a new theoretical model that captures dynamic aspects of the presentation layer of web applications and a testing methodology. However, integration of the server-side scripting, like JSP and Servlets, which can dynamically create user interfaces, is not included in this work.

Anneliese A. Andrews et al. proposed [2, 3] an approach to build hierarchies of Finite State Machines (FSMs) that model the subsystems of web-applications, and then generate test requirements as subsequence of states in the FSMs. This subsequence are then combined and redefined to form complete executable tests. The constraints are used to select a reduced set of inputs to reduce the state space explosion. In other related works, Christophe Meudec[13] on "automatic generation of software test cases from formal specifications" mentioned the need of extra data types to deal with the detail level of specification. Ali Mesbah et al.,[6] has proposed a state-flow model for handling UI requirements and tried to find an automated testing policy for AJAX applications.

3. Proposed framework

The proposed framework (Fig.1) incorporates different components like the SRS, the Analysis model, the verification model and the design specification. The first step in our approach is to build the analysis model, which is composed of formal specifications of the requirements at two different abstraction-levels, using VDM-SL. We consider that the functional requirements are specified in the SRS in conventional manner using plain text and the interface-requirements are represented by graphics, drawing or screen-shots. Then we create the level-1 VDM-SL from the SRS, which is nothing but the SRS written in a formal way using VDM-SL. Next, the level-1 specification is converted into level-2 specification using the technique specified in section 6. The interface-requirements are further explored while performing the conversion. The level-2 specification is less abstract and focused towards the design of the system from the software engineers' perspective, in contrast to level-1 specification, where the focus is on the clients' perspective of what the system will do. We consider the level-2 specification as the design specification, which could have been otherwise constructed by converting a design artifact (directly obtained from the SRS using conventional methods) in to a VDM-SL specification. However, the mapping technique between the level-2 specification and the design artifact is not considered in this paper. This level-2 specification illustrates the generic inputs and outputs in further details in the context of web based application development. We propose a set of data types and operations, both in level-1 and level-2, which helps in including the interface-requirements within the specifications. Next, we have defined a technique in section7 to create an FSM model from the level-2 specification. Finally, we create test cases using the FSM model and compare the results with the SRS in order to verify the design specification.

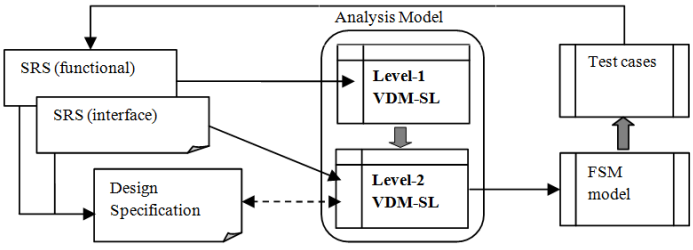

Fig. 1.: Framework for applying formal methods in Requirements

4. Formalizing interface-requirements in web-application

Designing an effective user interface to a complex information system is difficult, since it cannot be done in isolation from the design of the information system itself. A formal specification of the system must include a description of the interfaces and their interactive behavioral nature, in the requirements specification. This eliminates the hazards of separation of UI and underlying functionality[10]. Moreover, the popularity of internet has made the users so convenient with the web based software that the users are not restrained within the functional need, instead they often advise (even demand) on the presentation and view of the software, and specify the way of interaction with the software. In other words, the users decide how the user-interface-requirement should be specified [1]. Many of the cases of software development do not bother to formalize the software interface-requirements. Arguably, there is little need to do so, if the FR are properly separated from the interface-requirements. However, in the case of web-applications development, the user interface is the most visible part of the system and the portion to which the end-users can most easily relate. It is, therefore, common for the stakeholders to specify the system requirements by means of screens or controls in the UI. Consequently, web-applications could benefit from formalization of the interface-requirements[12]. In web-applications, the interface-requirements fit into the input and output components of the FR. Further, web-applications, being event-based application, some level of formalization could also be helpful in modelling the control flow of the application, which is not always apparent due to the stateless nature of HTTP [12]. The user's intervention is required to start a functional module and the user's interpretation of the result is required to complete the functional module. Considering all these, we propose some add-ons to VDM-SL in such a way that it should be able to define the interface-requirements along with the conventional FR with same efficiency. We proposed some VDM-SL data types and operations to cover the interface-requirements so that a certain degree of formalization could also be applied to denote input constraints as size limit, data type, selection from a specified set of choices, default values, etc.

We consider web-applications as collection web pages, which are either static pages that contain only HTML and client side executable code (e.g. JavaScript) and reside on a web server or dynamic pages that are generated as the result of the execution of server-scripts (e.g. Servlets) at the server. Dynamic web pages may contain a mixture of HTML source and sever side executable code (e.g. JSP), and are served by the application server. Other than web-pages, web-application may also have web services. They are primarily used as a means for communication, usually in XML, and not tied to users' interface or HTML[11]. Web-applications may also include active-web technologies like Java Applet, ActiveX, Flash object, and Multimedia components (Image, Audio, Video). If these objects are non-interactive we may consider them as a mere display pages; whereas, for interactive components we must consider them as a separate application working top of presentation layer of the web-application. Therefore, Active-web components and Web- services are outside the scope of the present work.

5. Proposed add-ons for two levels of VDM-SL specification

In traditional waterfall-style, formal modeling could be applied at the requirements specification level and then can be refined downward to the formal description at the design level. The first level involves translating the system requirements into formal notations in terms of the high-level entities used in the system, their properties, and interactions, without attempting any technical aspects of implementation. In the design-level specification, formal methods are used to describe the information on the objects to be included in the system, their properties and operations, and some details of how they work. Unlike the previous level, this specification gives at least a loose description about the relationship between the implementation and the formal model [12]. Table 1 and Table 2 represent add-ons for the level-1 specification whereas Table 3 and Table 4 depict the same for the level-2 specification.

5.1. Add-on for level-1 specification

Any web-based application can be seen as a collection of web pages where web pages are connected with links. Links can be established in two different ways: i) static links are html elements 'href' or 'form', ii) a dynamic link can be established by server-side 'redirection' script or by client-side form submission script. In the level-1

specification, both i) and ii) are represented by the data type LK. The navigation operator works only if there is an already established link between the web pages. The system may impose restrictions on the allowed data values of the input data. This can be realized at the time of binding of the basic data types of VDM-SL with the input data types. For example, b: IP::Z represents that b is an input type variable that takes only integer values. The system passes information back to user by writing on the visible section of the web page. In our approach, we simply assign the value to the variable of output data type.

Table 1: Level-1 data types

Data Types	Symbol	Declaration	Remarks
Web-Page	WP	p:WP	Represents any web page visible in single window frame in the browser
link	LK	l: LK	Represent the link through which we can navigate from one page to another
input	IP	b: IP	Represent how user enter value to the system
output	OP	d:OP	Represent how system shows information to user

Table 2: Level 1 operations

Operator	Symbol	Use	Return type	Remarks
Type binding	::	b:IP::Z	-	b is a input type variable that takes integer value
Include	$\sum$	P $\sum$ b	WP	The web page P contains the input variable b
Set-link	$\prod$	l $\prod$ P2	LK	The link l is connected with the page P2
Navigate	►	P1► P2	WP	The web page P1 navigates to the web page P2 [possible if at least one link of P2 exists in P1]

5.2. Add-on for level-2specification

Table 3: Level-2 data types

Data Types	Symbol	Declaration	Remarks and Assignments
Web-Page	WP	p:WP	Creates an web page
ServerForm	SF	f:SF	Carries data to server-end
ClientForm	CF	c:CF	Carries data to client-end
Hyperlink	HL	l: HL	l = “Kolkata” means the link is visible as “kolkata”
TextBox	TB	t: TB	b= “Hello Kolkata” means the textbox contains “Hello Kolkata”
CheckBox	CB	c: CB	c= “checked” means the Check box is ticked, the other possible value is “unchecked”
RadioButton	RB	r: RB	r= “on” means the Radio button is selected, the other possible value is “off”
Dropdown	DD	d: DD	d=<”kolkata”| “Delhi”> means the dropdown has two values
PushButton	PB	b:PB	b = “OK” means the label on the button is “OK”
SubmitButton	SB	s:SB	s= “Submit Data ” means the label on the submit button is “Submit Data”
MessageBox	MB	x: MB	x= “Hello Kolkata” means the text will be displayed in a message box
PageScan	PS	s:PS	Represents how input data are fetched and only one PS is allowed in a page
PageWrite	PW	p:PW	Represents how to write in a web page and only one PW is allowed in a page
ScriptedLink	SL	s:SL	Represents s as a client-post or server-redirection link

Table 4: Level-2 operations

Operator	Symbol	Use	Return type	Remarks
Write	Ω	pwΩ “error” where, pw:PW	WP	The text “error” will be displayed in the web page that contains pw
Retrieve-Value	⊥	P $\sum$ s where, s: PS s ⊥ f,t where, f: SF or, s ⊥ c, t where, c : CF	Token	The value inside the textbox t is now available to page P either by client form c or by server form f. Other than text box it can be any component among HL,TB,CB,RB,DB
Pass-Param	ς	l ς v1	Token	The value inside the variable v1 is now set to the link l and will be available to the linking page P after navigation
Submit	Θ	fΘp2	-	The server or client form will go to the web page p2 with or without data.
intervention	**intv**	**intv**<system>(*ActiveControl, ActiveEvent)*	-	Invoked by the system’s environment and described by a control and an event

In level-2, we extend the basic input and output data elements as the control-objects of web based applications. The users interact through the users' interface controls like textbox, radio button, check box, dropdown box, hyperlink, push button and submit button. The values passed by the user are made available to the client side by a container called 'client form' and at the server-side by 'server form'; hence, all the control-objects should be placed in either a server-form or a client form. In the level-2 specification, the data type HL represents the static link. The dynamic linking is established either by the *ScriptedLink* data type or by the *Submit* operator. Data values can be passed from one web page to another in two different ways. The implicit way is to put the input variables in a server-form and add a *SubmitButton* in the web page; all the variables available in the form will be available to the target page when the *Submit* operation is performed. The other way is to pass the parameters explicitly using the *pass-param* operation with the hyperlink. In order to handle the event based nature of the web based systems we propose an 'intervention' function. It is invoked by the system' environment whenever there is any user's intervention in the system. The reason could be client-side events like clicking button, submitting, selecting from the dropdown list, etc. or server-side events like redirecting, session expired, etc. The keyword *intv* is used to define an intervention function. The syntax is: *intv<system> (ActiveControl, ActiveEvent) {}*

It allows two parameters, the first one is *ActiveControl* type and the other is *ActiveEvent* type. The parameter-values are dynamically set by the system's environment based on the intervention type. For instance, if a submit button is clicked then the values are set to *SB* and *Click* respectively. A hierarchical structure of the proposed abstract syntaxes and their relationships, used in the two-leveled specification language, is shown in Fig. 2.

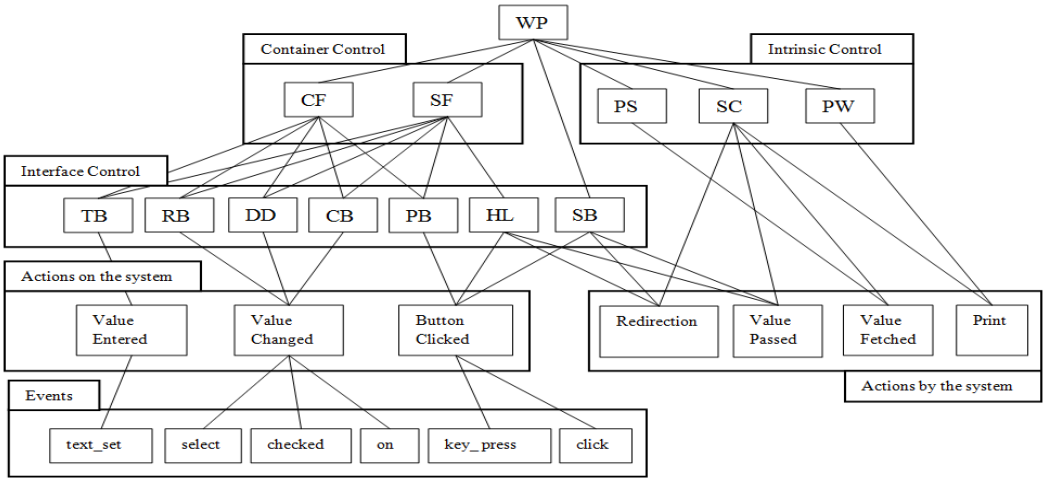

Fig. 2.:Hierarchical structure for the proposed syntaxes

6. The conversion-technique for level-1 specification into level- 2 specification

Definition1. Interface-Variable: The web control-objects are included in the VDM-SL specification as interface-variables. Interface-variables are included under state variables and any changes made to them results in a state change.

As stated earlier, the objective of the conversion of level-1 specification to level-2 specification is to produce a design specification from a requirement specification written formally.

Step I: All the inclusive-declaration of the operations are converted into exclusive-declaration
Step II: All the input and output controls are converted into corresponding web control-objects in accordance with the interface specification
Step III: Control-objects are added as interface variables in the state variable section.
Step III: PageScan and *PageWrite* objects are added to access and response to user data respectively.
Step IV: For all the navigations following a change in ext variable, implicit or explicit data passing is added

Step V: Applicable events are described by overriding the *intv* function.

7. The conversion-technique for level-2 specification into FSM model

Definition2. *Logical Page(LP):* a web page can be split into different logical pages; where each of the logical pages reflect either any change of value in its interface variable or change of state due to any action performed by/on the system.

Step I: Each of the Logical Pages represents a Node
Step II: Each of the events represents an edge between the source LP and the destination LP; it can also form self-loop.
Step III: Values of all the *ext* variable should be set before any state transition

8. A case study

Next we will explore the application of the proposed approach with help of a case study. We consider a system that should be able to convert the measure of length and weight from FPS unit to SI unit and the reverse.

8.1 Functional requirements:

FR1 The user should specify the physical quantity to be changed.
FR2 The user should specify the source unit
FR3 The user should specify the target unit
FR4 The user should enter the value
FR5 The user should click the "calculate" button
FR6 The result will be displayed

Due to the inherent ambiguity involved in the specifications written in natural languages, one SRS can lead to multiple interface specifications. In the above example, different designers can interpret the word 'specify' differently (Fig.3).

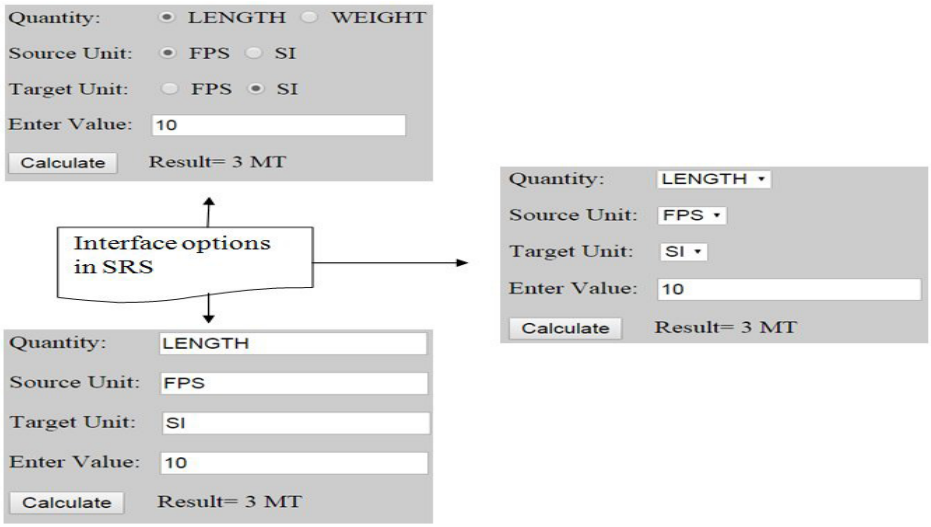

Fig.3: Multiple Design Choices

Therefore, an additional interface specification is often required in order to aviod the ambiguity. Let us consider that the client and end users agreed to have dropdown menu to give input for the first three requirements.

8.2 Interface-requirements

IR1. The interface should provide drop-down list, along with a label for the physical quantity to be changed.
IR2. The interface should provide drop-down list, along with a label for the source unit
IR3. The interface should provide drop-down list, along with a label for the target unit
IR4. The interface should provide textbox, along with a label to enter the value
IR5. The interface should provide the "calculate" button
IR6. The result will be displayed as plain text

8.3 VDM-SL level-1 specification

types
QuanType = <WEIGHT>|<LENGTH>
UnitType= <FPS>|<SI>
state *Convert_Calc* of
Quan : *IP::QuanType*
Unit1 : IP ::UnitType
Unit2 :IP:: UnitType
InpVal :IP::Z
Result : OP::Z
P:WP
init mk-*Convert_Calc*(*Quan* : *IP::QuanType, Unit1 : IP ::UnitType, Unit2 :IP:: UnitType, InpVal :IP::Z, Result : OP::Z, P:WP)* Δ
P = $\overline{P}\sum$ (quan,unit1,unit2,inpval,op)
end

operations
convertUnit()
ext wr *Result : OP*
rd *Quan : IP::QuanType*
Unit1 : IP ::UnitType
Unit2 :IP:: UnitType
InpVal :IP::Z
Result : OP::Z

Pre *unit1≠unit2*
Post
*Quan=(LENGTH ^ unit1=FPS ^ unit2= SI ^ result=inpval * 0.3)*
*∨ Quan=(LENGTH ^ unit1=SI ^ unit2= FPS ^ result=inpval * 3.2)*
*∨Quan=(WEIGHT ^ unit1=FPS ^ unit2= SI ^ result=inpval * 0.4)*
*∨ Quan=(WEIGHT ^ unit1=SI ^ unit2= FPS ^ result=inpval * 2.2)*

The level-1 specification specifies the web-application in an abstract way by introducing only three major data types: *input*, *output* and *web-page*. All input and output variables are included in the web-page variable, which work as a container variable. In this particular example, one web page is sufficient to represent the system so we have not used the link data type and its operations. The proposed VDM-SL level-1 uses a generic approach to specify requirements of a web-based application at a higher level of abstraction. Therefore, the conventional functional requirements, at an abstract-level and without much detail of the interface requirements, can also be included in the model.

8.4 VDM-SL level-2 specification

types
QuanType = <WEIGHT>|<LENGTH>
UnitType= <FPS>|<SI>
state *SystemName*of
quan : *DD::QuanType*
unit1 : *DD::UnitType*
unit2 : *DD::UnitType*
InpVal : TB::Z
Result : token
q: token
u1: token
u2: token
calc : PB
P : WP
f:CF
S:PS
pw:PW

init mk-*Convert_Calc (quan : DD:: QuanType, f:CF, unit1 : DD:: UnitType,unit2 : DD:: UnitType, InpVal : TB:: Z,Result : Z,P : WP, ps:PS, pw:PW, calc : PB)*Δ
P=$\overline{P}$ ∑ pw
P=$\overline{P}$ ∑f
pwΩ "quantity:"
P= $\overline{P}$ ∑quan
pwΩ "given unit:"
P= $\overline{P}$ ∑unit1
pwΩ "target unit:"
P=$\overline{P}$ ∑unit2
pw Ω "enter the value:"
P=$\overline{P}$ ∑(InpVal,calc)
Quan=<LENGTH>
Unit1=<FPS>
Unit2=<SI>
Result=nil
end
operations
convertUnit() Δ
ext wr Result : Z
rd quan : QuanType
unit1 : UnitType
unit2 : UnitType
InpVal : Z
q= P⊥c,quan
u1= P⊥c,unit1
u2= P⊥c,unit2
if *q= 'LENGTH'* and *u1= 'FPS'* and *u2= 'SI'* then
*result=inpval * 0.3*
else
if *q= 'LENGTH'* and *u1='SI'* an
u2= 'FPS' then
*result=inpval * 3.2*
else
if *q= 'LENGTH'* and *u1=FPS* a
u2= 'SI' then
*result=inpval * 0.4*
else
if *q= 'LENGTH'* and *u1='SI'* and *u2= 'FPS'* then
*result=inpval * 2.2*
end if
intv *UserAthentication (X:*

Δif *X= calc* and *Y= CLICK*
then
convertUnit()
pwΩ "Result="Ω result
end if
if *X= quan* and *Y= change* then
if *quan≠* $\overline{\text{quan}}$
then
quan=$\overline{quan}$
endif
if *X= unit1* and *Y= change* then
if *unit1* ≠ $\overline{unit1}$ then
unit1=$\overline{unit1}$
endif
if *X= unit2* and *Y= change* then
if *unit1* ≠ $\overline{unit2}$ and *unit2* ≠ $\overline{unit2}$ then
unit2=unit2
endif
if *X= InpVal* and *Y= change*
then
if *InpVal≠* $\overline{InpVal}$ then
InpVal =$\overline{InpVal}$

ActiveControl, Y : ActiveEvent) endif

The level-2 of specification is used to describe the interface-requirements more precisely while keeping the FR intact. We have included only one *ClientForm* form as the container for the input and output controls, as there is no need for server side data processing. We have converted the first three input types as *dropdown* object at the level-2. As they are included after the inclusion of the client form, they will be automatically assigned under it. The main operation *convertUnit()* is invoked under the *click* event of the button *calc*.

8.5 The FSM model:

From the above level-2 specification, we identify six logical web pages. We observed that there might be behavioral changes of the system due to six defined actions, each of which will produce a state in the FSM model.

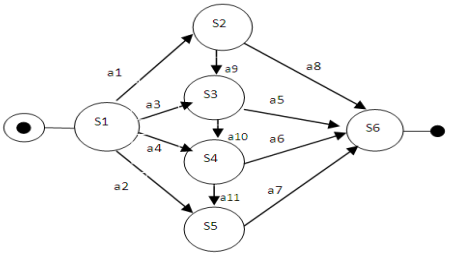

Fig.4: FSM model for testing

Table 5: Action performed on logical pages

Action Performed	State//Logical pages
Page loaded	S1
Quantity changed	S2
Unit1 changed	S3
Unit2 changed	S4
Value entered	S5
Button clicked	S6

The testing model (Fig.4) is based on the concept of logical pages and their participation on the events; the logical pages represent the states and the events represent the transitions. Some of the actions performed on the system will cause a state change of the system; table 5 presents a list of such states. The actions are generic concepts and ultimately realized by the events of the system. For example, the action 'Button clicked' can be realized by an event *calc_click()*. Every event can be further broken into a set of sub events; like button press can be broken into key press and key up events or setting the value in a textbox can be decomposed as a series of character entry events. However, such sub events generally do not reflect any behavioral change in the state of the system; moreover if considered in FSM, they may lead to the state-space-explosion problem. Hence, we have concentrated only on events, not sub-events. Certain events like change in *DropDown*, check/uncheck *CheckBox*, on/off *RadioButton* and setting values in *Textbox,* automatically assign values to the interface variables. On the other hand, events like button click, do not directly assign values to any interface variables, instead they can set values to the other state variables like passing parameters. Table 6 shows the list of events that perform the transition of states and the corresponding state variables they change (if any).

Table 6: State-transition

Transition	Variable set	Event
a1	quan	quan_change
a2	inpval	InpVal_text_set
a3	unit1	unit1_change
a4	unit2	unit2_change
a5	-	calc_click
a6	-	calc_click
a7	-	calc_click
a8	-	calc_click
a9	unit1	unit1_change
a10	unit2	unit2_change
a11	inpval	InpVal_text_set

Table 7: Test sequences

Test ID	Start, Terminate	Sequence	Expected behaviour
T1	S1,S6	a1→ a9→ a10→ a11→ a7	Correct result
T2	S1,S6	a1→ a9→ a5	Error/No result
T3	S1,S6	a1→ a9→ a10→a6	Error/No result
T4	S1,S6	a1→ a8	Error/No result
T55	S1,S6	a3→ a5	Error/No result
T6	S1,S6	a3→ a10→ a6	Error/No result
T7	S1,S6	a3→a10→ a11→ a7	Correct result
T8	S1,S6	a4→ a6	Error/No result
T9	S1,S6	a4→ a11→ a7	Correct result
T10	S1,S6	a2→ a7	Correct result

8.6 Test case generation:

Test cases can be constructed easily from the above FSM model. The complete path of a test case should have S1 as start node and S6 as end node. It is obvious that only few of the user's action sequences will turn out the correct result, others will produce error or no result. Table 7 shows all the possible sequences by the users and the expected behaviours. As stated in the technique at *sec viii*, every state change requires to set the *ext* variable before transition.

For example, Test T1 requires the following actions on variables: [see table 7]
Set *quan*, Set *unit1*, Set *unit2*, Set *inpval*, click *button*
As all the test cases start from S1 (page load), we must assume default value for the ext variables at the beginning and also when they are by-passed in the test sequence.
For example, Test T9 requires the following actions on variables to form a valid test path:
[*quan*=<default>], [*unit1*=<default>], Set *unit2*, Set *inpval*, click *button*

9 Conclusion

In this work, we have introduced a framework to perform formal representation and verification on the requirements of web-applications. We have proposed some add-ons to the VDM-SL to model the FR and the interface-requirements in an integrated way. In our approach, we have used the specification language in two different levels of abstractions; the level-1 specification is suitable for predominant FR and the interface-requirements being stated in a more abstract way. In contrast, the level-2 specification is suitable for compound requirements, i.e., interface-requirements being stated in a detailed way, usually specified in a separate artefact, keeping the FR intact. In our proposed approach, we have first created the level-1 specification and then converted it into level-2 specification, using a proposed conversion technique. Then we have demonstrated another technique to develop a FSM model from the level-2 specification. Finally, the test cases have been generated from the FSM model. In our case study, requirements that demand use of multiple web pages and server side data processing are not considered for the sake of simplicity, though the proposed model is capable of handling such requirements. However, the other NFR like communication-requirements and data-requirements require more add-ons to the model, which we consider as a future scope of this work.

References

1. Sengupta S, Dasgupta R. Integration of Functional and Interface Requirements of an Web Based Software: A VDM Based Formal Approach. *Proceedings of IASTED International Conference on Software Engineering* 2013;2013.DOI: 10.2316/P.2013.796-017
2. Andrews A, Offutt J, Alexander R T. Testing web applications by modeling with FSMs. *Software & Systems Modelling, 4(3)*;2005. p326-345.
3. Andrews A, Offutt J, Dyreson C, Mallery C J, Jerath K, Alexander R. Scalability issues with using FSMWeb to test web applications. *Information and Software Technology*, 52(1);2010. p52-66.
4. Müller A. VDM—The Vienna Development Method. Bachelor thesis in" Formal Methods in Software Engineering", *Research Institute for Symbolic Computation (RISC)*, Johannes Kepler University Linz, Austria;2009.
5. Offutt J, Wu Y. Modelling presentation layers of web applications for testing. *Software & Systems Modelling,* 9(2); 2010.p257-280.
6. Mesbah A, van Deursen A, Roest D. Invariant-based automatic testing of modern web applications. *Software Engineering, IEEE Transactions on*, 38(1), 2012. p35-53.
7. Atterer R. Automatic test data generation from VDM-SL specifications. *The Queen's University of Belfast*; 2000.
8. Liu S. A formal requirements specification method based on data flow analysis. *Journal of Systems and Software*, 21(2); 1993.p141-149.
9. IEEE Computer Society. Software Engineering Standards Committee, IEEE-SA Standards Board. IEEE Recommended Practice for Software Requirements Specifications. *Institute of Electrical and Electronics Engineers*; 1998.
10. Bowen J A. Formal models and refinement for graphical user interface design. *Doctoral dissertation, The University of Waikato*; 2008.
11. Alalfi M H, Cordy J R, Dean T R. Modelling methods for web application verification and testing: state of the art. Software Testing, Verification and Reliability, 19(4);2009. p265-296.
12. Geer P A. Formal Methods In Practice: Analysis and Application of Formal Modelling To Information Systems, *Doctoral dissertation, State University of New York Institute of Technology*; 2011.
13. Meudec C. Automatic generation of software test cases from formal specifications, *Doctoral dissertation, The Queen's University*;1998.
14. Mernik M, Heering J, Sloane A M. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)*, 37(4); 2005, p316-344.